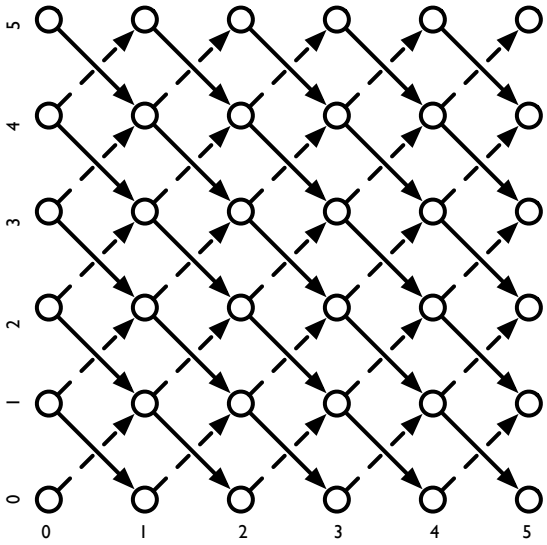
ECE 468
Problem Set 8: Dependence analysis

1. Consider the following code:

```
for (int i = 0; i < 6; i++) {
   for (int j = 0; j < 6; j++) {
      A[i][j] = A[i - 1][j + 1] + A[i + 1][j + 1];
   }
}
```

   (a) Draw the iteration space graph for this loop nest. Use solid arrows for flow dependences, dashed arrows for anti-dependences and dotted arrows for output dependences.

   Answer:



   (b) Give the distance and direction vectors for each type of dependence.

   Answer: Flow: $(1, -1)$ or $(+, -)$. Anti: $(1, 1)$ or $(+, +)$

   (c) Can we perform loop interchange on this loop? Why or why not?

   Answer: No, we cannot. The flow dependence would turn into an anti dependence if we performed loop interchange.

2. Consider the following code:

```
for (int i = 0; i < N; i++) {
   for (int j = 0; j < N; j++) {
      A[j][i] = A[j - 1][i - 1] + B[i][j];
      A[j+1][i] = B[i + 1][j - 1];
   }
}
```

(a) Give the distance and direction vectors for the dependences in this loop. Indicate which array the dependences are on.

Answer: We only read from `B`, so there are no dependences on it. The dependences on A are:

- Flow: $(1, 1), (1, 2)$. Both of these are $(+, +)$ vectors. Note that the distance vector is $(1, 2)$, not $(2, 1)$. This is because the i loop is the outer loop, so its component of the vector must be listed first.
- Output: $(0, 1)$, which is a $(0, +)$ vector.

(b) Is loop interchange legal for this loop? Why or why not?

Answer: Loop interchange is legal because all of the dependences in the loop would be preserved.

(c) Assuming both the `A` and `B` arrays are stored in row-major order, do you expect loop interchange to be beneficial for this code? Why or why not? Note that in modern processors, there are essentially no latency penalties for cache misses that occur during stores; it is mostly only cache misses during loads that affect performance.

Answer: In the original loop, `A` is accessed in column major order, and `B` is accessed in row major order. This means that accesses to `B` enjoy spatial locality, but accesses to `A` do not. In the interchanged loop, the opposite situation would occur. Because there are more reads to `B` than `A` per iteration, we would prefer to get locality in `B`, so we should not perform loop interchange.